Web Components

HSR

Herbst 2020

Alex Kühne

BA Web Development & Design
Eidg. FA Informatiker / Applikationsentwickler

swisscom

SIX

Swiss Re

Tages-Anzeiger

MIGROS

Web Components

- wer hat schon davon gehört?
- wer hat damit gearbeitet?

Agenda

- was sind Web Components?
- W3C Standard
- Scoped CSS
- Libraries
- *Zwischendurch: Praxis*

Was sind Web Components?

…was sind überhaupt „Components“?

„Components“

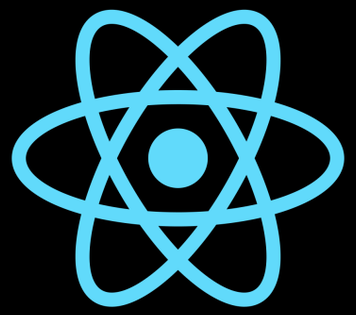
…sind ein Pattern für Wiederverwendbarkeit
in der Softwarearchitektur

„Components“

…im Browser angewandt:

```
<my-button>

<app-navigation>

<custom-tabs>
```

Alle Major Frameworks implementieren jetzt „Components“:


Angular

```
<my-user></my-user>
```

Angular

```
<my-user>
  <span>{{user.name}}</span>
</my-user>
```

Angular

```
<my-user
  size="large"
>
  <span>{{user.name}}</span>
</my-user>
```

Angular

```
<my-user
  size="large"
  *ngFor="let user of users"
>
  <span>{{user.name}}</span>
</my-user>
```

Angular

```
<my-user
  size="large"
  *ngFor="let user of users"
  (onClick)="onUserClick()"
>
  <span>{{user.name}}</span>
</my-user>
```

Angular Component

```
class MyUser {
}
```

Angular Component

```
@Component()
class MyUser {
}
```

Angular Component

```
@Component({
  selector: "my-user"
})
class MyUser {
}
```

Angular Component

```
@Component({
  selector: "my-user",
  templateUrl: "my-user.html"
})
class MyUser {
}
```

Angular Component

```
@Component({
  selector: "my-user",
  templateUrl: "my-user.html"
})
class MyUser {
  @Input() size: string;
}
```

Angular Component

```
@Component({
  selector: "my-user",
  templateUrl: "my-user.html"
})
class MyUser {
  @Input() size: string;

  onUserClick() {
    // …
  }
}
```

React

```
users.map(user => {});
```

React

```
users.map(user =>
  <MyUser></MyUser>
);
```

React

```
users.map(user =>
  <MyUser>
    <span>{user.name}</span>
  </MyUser>
);
```

React

```
users.map(user =>
  <MyUser
    size="large"
  >
    <span>{user.name}</span>
  </MyUser>
);
```

React

```
users.map(user =>
  <MyUser
    size="large"
    onClick={(e) => onUserClick(e)}
  >
    <span>{user.name}</span>
  </MyUser>
);
```

React Component

```
class MyUser extends React.Component {
}
```

React Component

```
class MyUser extends React.Component {
  onUserClick() {
    // …
  }
}
```

React Component

```
class MyUser extends React.Component {
  onUserClick() {
    // …
  }

  render() {
    // …
  }
}
```

Vue

```
<my-user></my-user>
```

Vue

```
<my-user>
  <span>{{user.name}}</span>
</my-user>
```

Vue

```
<my-user
  :size="large"
>
  <span>{{user.name}}</span>
</my-user>
```

Vue

```
<my-user
  :size="large"
  v-for="user in users"
>
  <span>{{user.name}}</span>
</my-user>
```

Vue

```
<my-user
  :size="large"
  v-for="user in users"
  v-on:click="onUserClick"
>
  <span>{{user.name}}</span>
</my-user>
```

Vue Component

```
Vue.component("my-user");
```

Vue Component

```
Vue.component("my-user", {
  template: "#my-user"
});
```

Vue Component

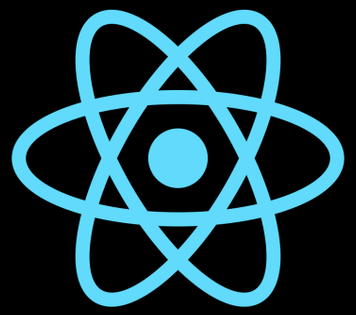
```
Vue.component("my-user", {
  template: "#my-user",
  props: {
    size: Number
  }
});
```

Vue Component

```
Vue.component("my-user", {
  template: "#my-user",
  props: {
    size: Number
  },
  methods: {
    onUserClick: () => {
      // …
    }
  }
});
```

Fazit

Gleiches Prinzip bei allen:


Components Vorteile?

- wiederverwendbar
- verschachtelbar
- testbar
- abgrenzbar (Ökosystem)

Wäre es nicht toll…

…wenn man dazu kein Framework benötigte?

Web Components

Was sind Web Components?

Web Components sind
die <u>nativen</u> Components des Browsers!

Web Components

Browserkompatibilität Herbst 2020

- Chrome
- Firefox
- Safari / iOS Safari
- Edge (ab 79, mit Chromium)
- IE11

Web Components

```
<my-user></my-user>
```

Web Components

```
<my-user
  my-attr="my-value"
>
</my-user>
```

Web Components

```
<my-user
  my-attr="my-value"
>
  <span>My content</span>
</my-user>
```

Web Components

```
<my-user
  my-attr="my-value"
>
  <span>My content</span>
</my-user>
```

- voll nativ - ohne Framework
- 2011 erstmals vorgestellt
- neuer W3C Standard

Web Components

Neuer W3C Standard?

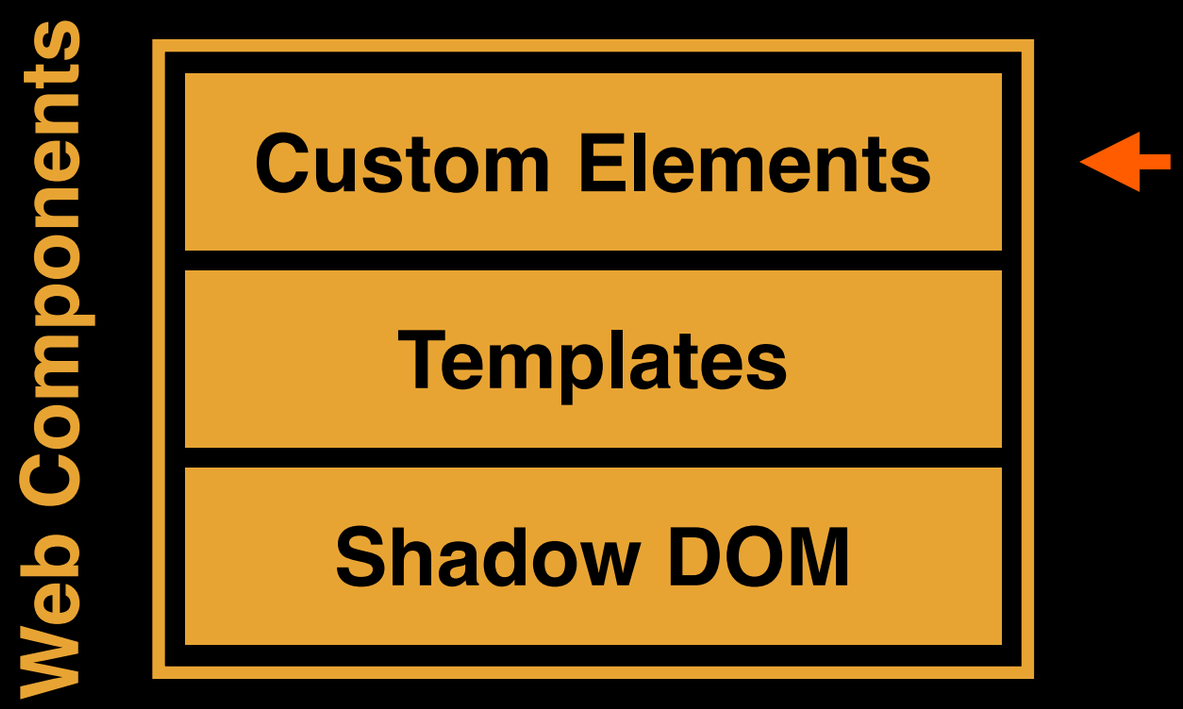
Web Components

~~Neuer~~ W3C Standard

Web Components

Drei neue!

~~Neuer~~ W3C Standards



Custom Elements

```
<my-user></my-user>
```

Custom Elements

```
<my-user></my-user>
```

Bindestriche müssen vorhanden sein!

Tags müssen immer geschlossen sein!

Custom Elements

```
class MyUser extends HTMLElement {
}
```

Custom Elements

```
class MyUser extends HTMLElement {
  constructor() {
    super();
  }
}
```

Custom Elements

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    this.innerHTML = `…`;
  }
}
```

Custom Elements

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    this.innerHTML = `
      <span>First name</span>
      <span>Last name</span>
    `;
  }
}
```

Custom Elements

```
customElements.define("my-user", MyUser);
```

Custom Elements

First name Last name

```
▼ <my-user>
    <span>First name</span>
    <span>Last name</span>
  </my-user>
```

Custom Elements

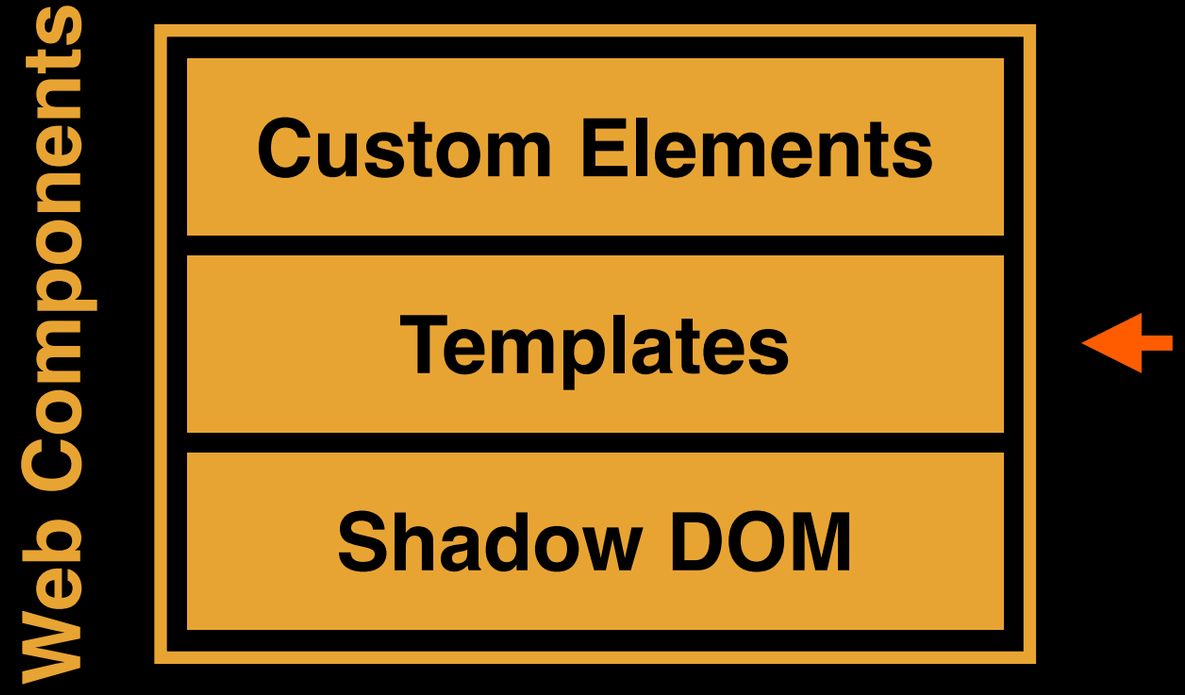
Template statt String?

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    this.innerHTML = `
      <span>First name</span>
      <span>Last name</span>
    `;
  }
}
```

Web Components

Drei neue!

~~Neuer~~ W3C Standards



Templates

```
<body></body>
```

Templates

```
<body>
  <h1>My Document</h1>
</body>
```

Templates

```
<body>
  <h1>My Document</h1>

  <template></template>
</body>
```

Templates

```
<body>
  <h1>My Document</h1>

  <template id="my-template">
  </template>
</body>
```

Templates

```
<body>
  <h1>My Document</h1>

  <template id="my-template">
    <span>First name</span>
    <span>Last name</span>
  </template>
</body>
```

Templates


My Document
?
<body>
<h1>My Document</h1>
<template id="my-template">
#document-fragment
</template>
</body>

Templates

#document-fragment?

Wie das **#document** Objekt,
allerdings nicht direkter Teil davon.

Mutationen daran erzwingen keinen kompletten Repaint.

** **#document-fragment** an sich ist nicht neu,*
*ist aber der Schüssel zum **<template>**.*

Templates

Template statt String?

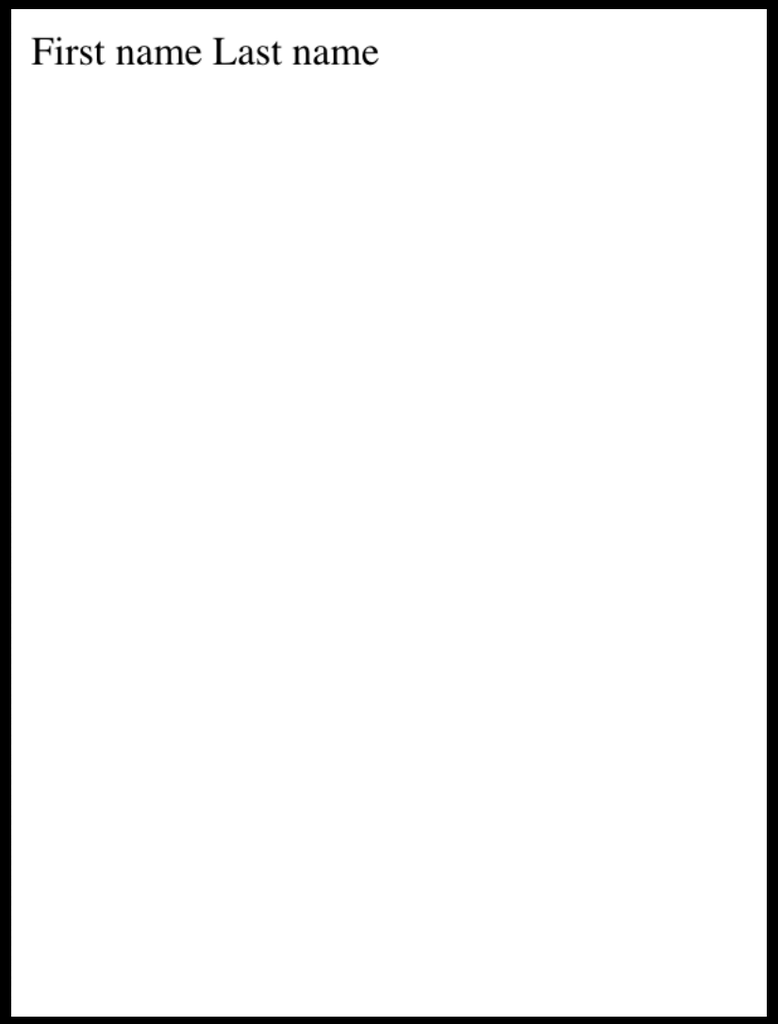
```
class MyUser extends HTMLElement {
  constructor() {
    super();
    this.innerHTML = `
      <span>First name</span>
      <span>Last name</span>
    `;
  }
}
```

Templates

Template statt String?

```
class MyUser extends HTMLElement {
  constructor() {
    super();
    this.innerHTML = document
      .querySelector("#my-template")
      .innerHTML;
  }
}
```

Templates



Templates

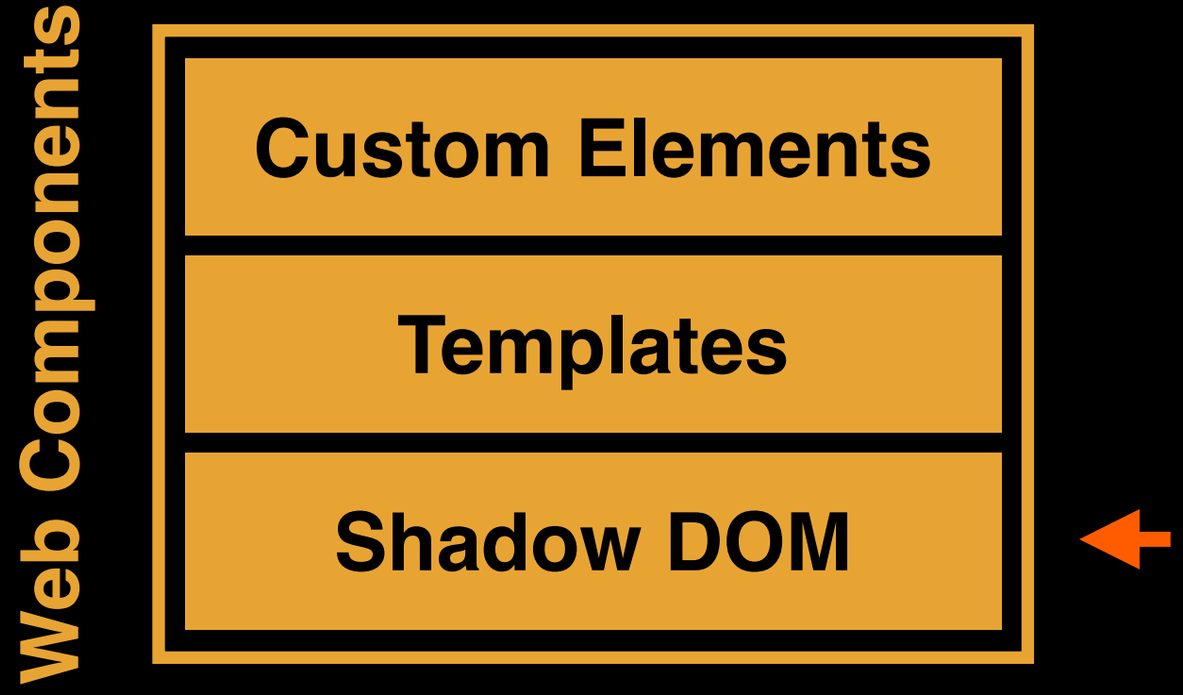
- Vorteil der Wiederverwendbarkeit
- Vorteil der Performance (Document-Fragment)

- keine Template-Engine mit Variablen, Loops, if/else, …
- kein Rendering wie z.B. Virtual DOM
- für den Alltag zu low-level, Libraries hier bieten mehr
- Übungen aus Einfachheit mit Strings statt Templates

Web Components

Drei neue!

~~Neuer~~ W3C Standards

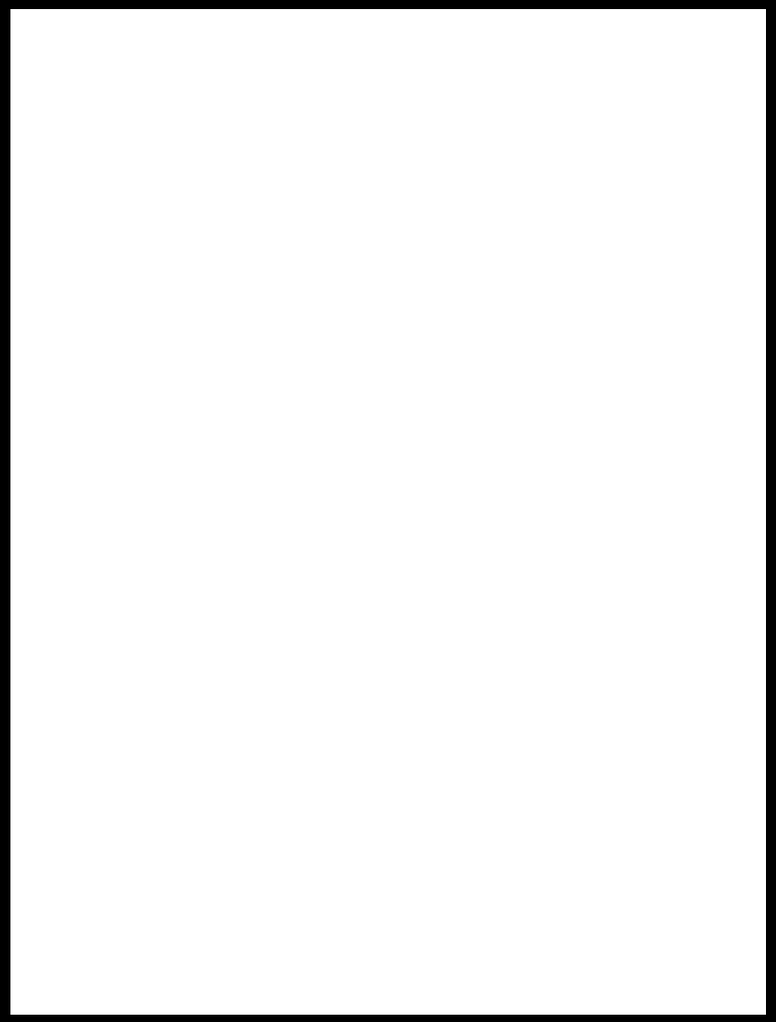


Shadow DOM

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    this.innerHTML = `
        <span>First name</span>
        <span>Last name</span>
    `;
  }
}
```

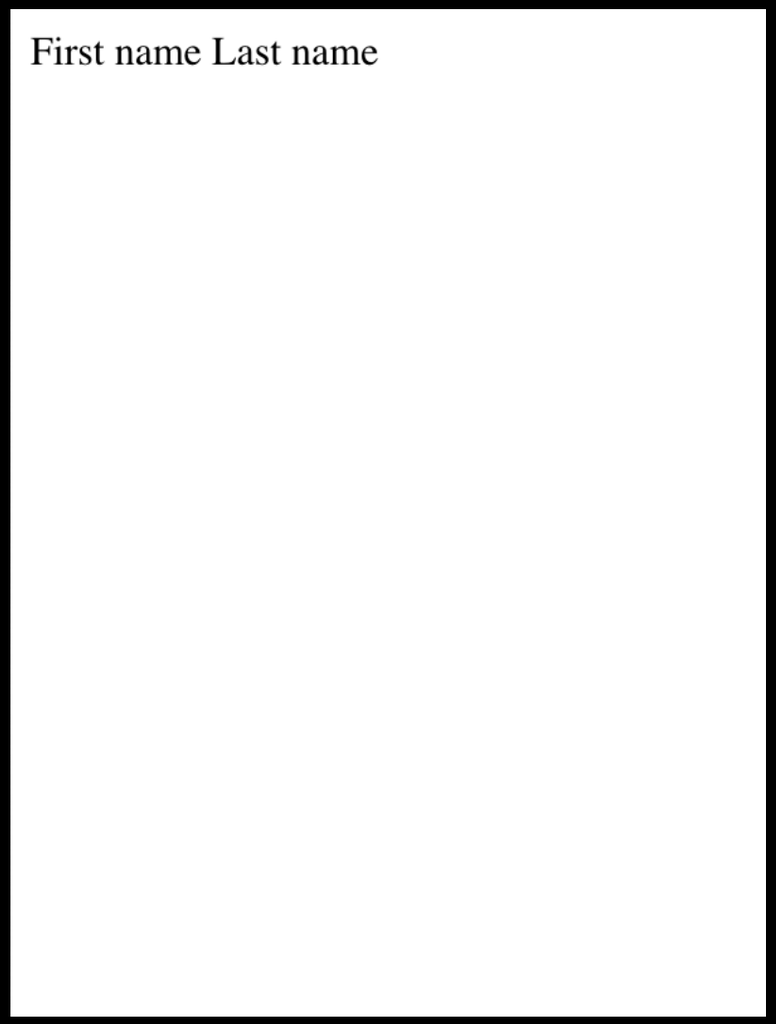
Shadow DOM

```
<body></body>
```



Shadow DOM

```
<body>
  <my-user></my-user>
</body>
```



Shadow DOM

```
<body>
  <style>
    span {
      color: brown;
    }
  </style>

  <my-user></my-user>
</body>
```

First name Last name

```
▼<my-user>
    <span>First name</span>
    <span>Last name</span>
  </my-user>
```

Shadow DOM

Shadow DOM

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    this.innerHTML = `
        <span>First name</span>
        <span>Last name</span>
    `;
  }
}
```

Shadow DOM

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    const shadowRoot = this
      .attachShadow({ mode: "open" });

    this.innerHTML = `
      <span>First name</span>
      <span>Last name</span>
    `;
  }
}
```

Shadow DOM

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    const shadowRoot = this
      .attachShadow({ mode: "open" });

    this.innerHTML = `
      <span>First name</span>
      <span>Last name</span>
    `;
  }
}
```

Shadow DOM

```
class MyUser extends HTMLElement {
  constructor() {
    super();

    const shadowRoot = this
      .attachShadow({ mode: "open" });

    shadowRoot.innerHTML = `
      <span>First name</span>
      <span>Last name</span>
    `;
  }
}
```

Shadow DOM

```
<body>
  <style>
    span {
      color: brown;
    }
  </style>

  <my-user></my-user>
</body>
```

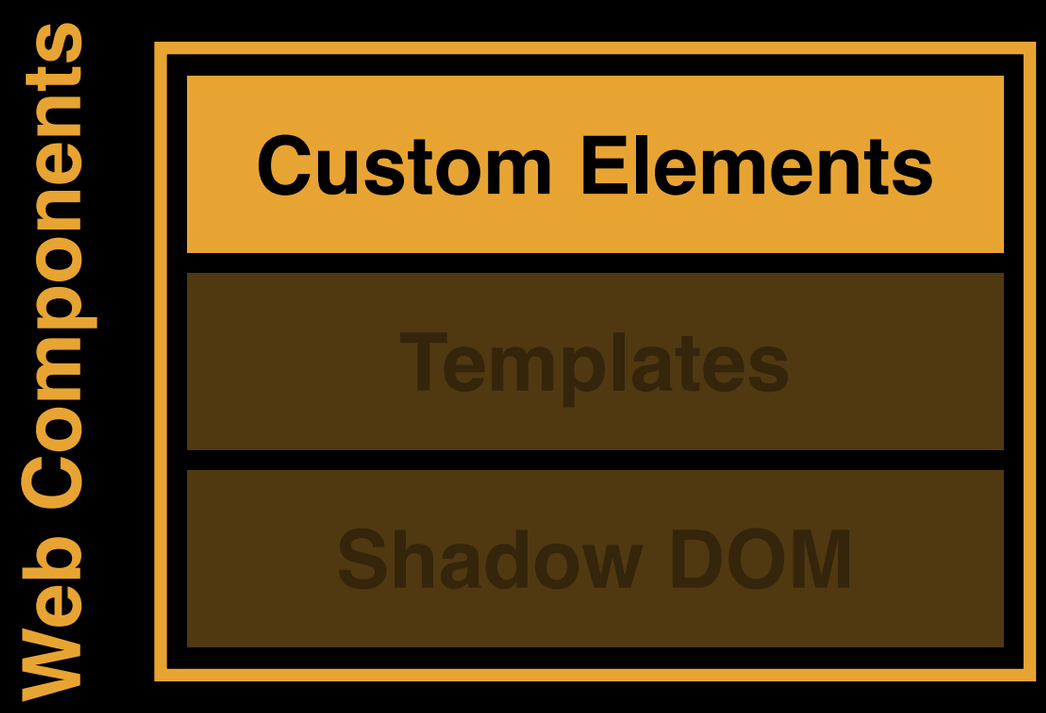
First name Last name

```
▼<my-user> == $0
  ▼#shadow-root (open)
      <span>First name</span>
      <span>Last name</span>
  </my-user>
```

Web Components

Das waren alle drei Standards.

Um eine Web Component zu bauen,
sind jedoch nicht alle erforderlich:



Web Components

vs. ?

Braucht es überhaupt noch ein Framework?

…es kommt drauf an!

Web Components

Web Components…

- „einfach“ Elemente erstellen und überall verwenden
- als UI Library für alle Frameworks
- wo Scoping (shadow-root) nötig ist

Frameworks…

- Router, Rendering, Store, …
- Tooling wie Dev Server, CLIs, …
- …können auch Web Components!

Praxisübung

Vorlage

```
<!DOCTYPE html>
<html>
  <body></body>
</html>
```

Praxisübung

Vorlage

```
<!DOCTYPE html>
<html>
  <body>
    <my-component></my-component>
  </body>
</html>
```

Praxisübung

Vorlage

```
<!DOCTYPE html>
<html>
  <body>
    <my-component></my-component>

    <script></script>
  </body>
</html>
```

Praxisübung

Vorlage

```
<!DOCTYPE html>
<html>
  <body>
    <my-component></my-component>

    <script>
      class MyComponent extends HTMLElement {
      }
    </script>
  </body>
</html>
```

Praxisübung

Vorlage

```
<!DOCTYPE html>
<html>
  <body>
    <my-component></my-component>

    <script>
      class MyComponent extends HTMLElement {
      }

      customElements.define("my-component",
        MyComponent);
    </script>
  </body>
</html>
```

Praxisübung

`<my-counter>`

für jede Übung!

```
$ npm install -g browser-sync
```

Ordner `webcomponents` erstellen & öffnen in IDE

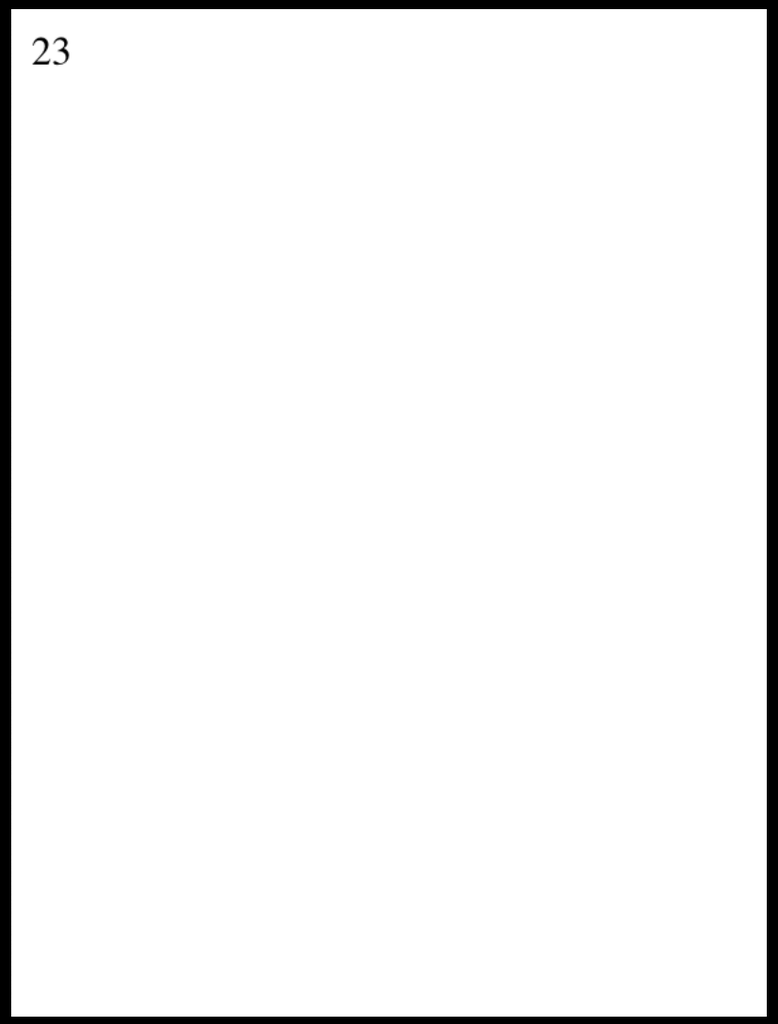
`<Übungsname>.html` gemäss Vorlage anlegen

```
$ browser-sync start --server
```

Browser: `localhost:3000/<Übungsname>.html`

Praxisübung

`<my-counter>`



Praxisübung

```
<my-counter>

class MyCounter extends HTMLElement {
  constructor() {
    super();
    const shadowRoot = …

      ?

  }
}
```

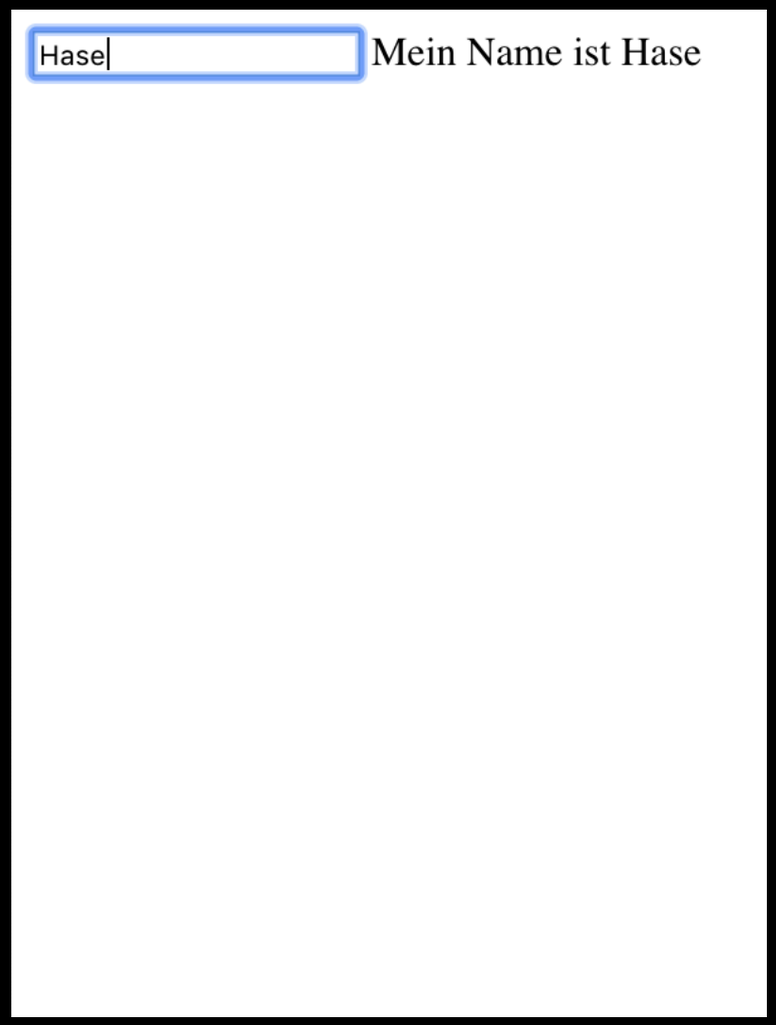
Praxisübung

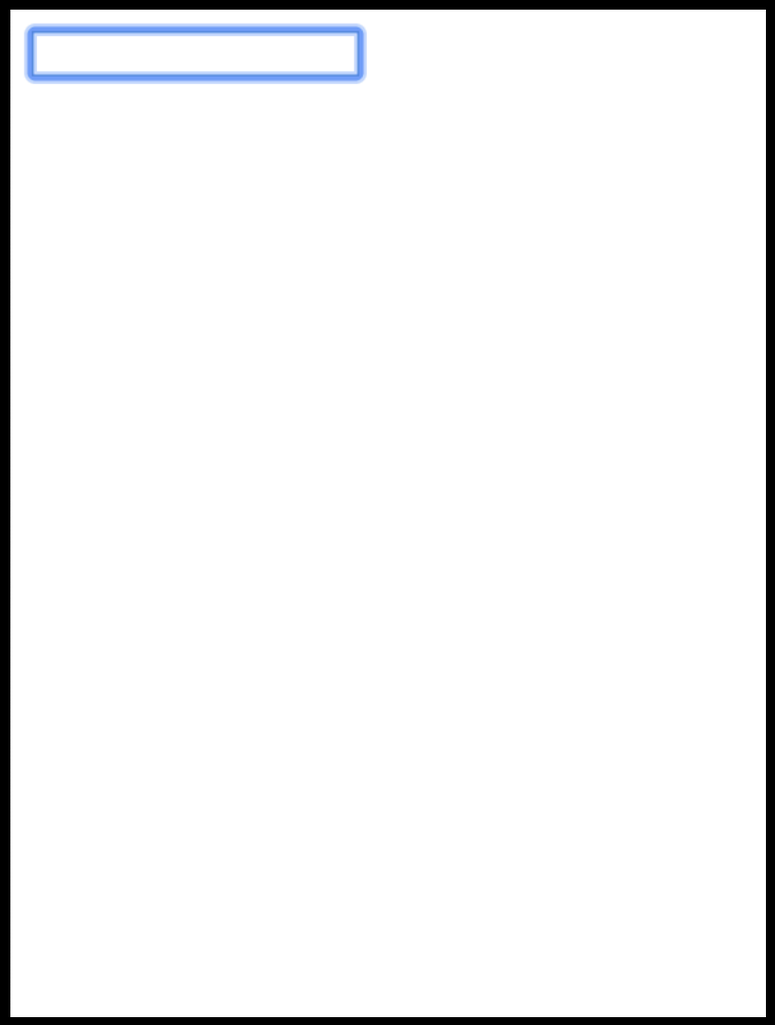
<my-counter>

```
class MyCounter extends HTMLElement {
  constructor() {
    super();
    const shadowRoot = …

    let counter = 0;

    setInterval(() => {
      counter++;
      shadowRoot.innerHTML = counter;
    }, 1000);
  }
}
```

Praxisübung

`<my-name>`





Praxisübung

`<my-name>`

Tipps

```
// .querySelector() auf shadow-root
shadowRoot.querySelector("input");

// Event Listener
input.addEventListener("keyup", (e) => {
  // e.target.value
});
```

Templating

`<slot>`

`<my-select></my-select>`

Templating

```
<slot>
```

```
<my-select>
  <div>Option 1</div>
  <div>Option 2</div>
</my-select>
```

Templating

```
<slot>

class MySelect extends HTMLElement {
  constructor() {
    super();

    const shadowRoot = …
  }
}
```

Templating

<slot>

```
class MySelect extends HTMLElement {
  constructor() {
    super();

    const shadowRoot = …

    shadowRoot.innerHTML = `
      <div class="wrapper"></div>
    `;
  }
}
```

Templating

<slot>

```
class MySelect extends HTMLElement {
  constructor() {
    super();

    const shadowRoot = …

    shadowRoot.innerHTML = `
      <div class="wrapper">
        <slot></slot>
      </div>
    `;
  }
}
```

Templating

<slot>

```
▼<my-select>
  ▼#shadow-root (open)
    ▼<div class="wrapper">
      ▼<slot>
          ↳ <div>
          ↳ <div>
        </slot>
      </div>
    <div>Option 1</div>
    <div>Option 2</div>
  </my-select>
```

Templating

if/else

```
shadowRoot.innerHTML = isLoggedIn
  ? "<logout-button></logout-button>"
  : "<login-button></login-button>"
;
```

Templating

loop

```
for (user of users) {
  shadowRoot.innerHTML += `
    <div>${user}</div>
  `;
}
```

Props

```
<my-component my-attr="my-value">
```

Props

```
class MyComponent extends HTMLElement {
  constructor() {
    super();

    if (this.hasAttribute("my-attr") {
      // this.getAttribute("my-attr");
      // this.setAttribute("my-attr", …);
    }
  }
}
```

Lifecycle

```
class MyComponent extends HTMLElement {
  constructor() {}
}
```

Lifecycle

```
class MyComponent extends HTMLElement {
  constructor() {}

  connectedCallback() { /* im DOM! */ }
}
```

Lifecycle

```
class MyComponent extends HTMLElement {
  constructor() {}

  connectedCallback() { /* im DOM! */ }

  disconnectedCallback() { /* cleanup */ }
}
```

Lifecycle

```
class MyComponent extends HTMLElement {
  constructor() {}

  connectedCallback() { /* im DOM! */ }

  disconnectedCallback() { /* cleanup */ }

  adoptedCallback() { /* other fragment */ }
}
```

Lifecycle

```
class MyComponent extends HTMLElement {
  constructor() {}

  connectedCallback() { /* im DOM! */ }

  disconnectedCallback() { /* cleanup */ }

  adoptedCallback() { /* fragment */ }

  attributeChangedCallback() { /* … */ }
}
```

siehe Code-Beispiel

Scoped CSS

```
class MyComponent extends HTMLElement {
  constructor() {
    super();
    const shadowRoot = …

    shadowRoot.innerHTML = `
      <span>My Component</span>
    `;
  }
}
```

Scoped CSS

My Component

```
<my-component></my-component>

<style>
  span {
    font-weight: bold;
  }
</style>
```

Scoped CSS

```
class MyComponent extends HTMLElement {
  constructor() {
    super();
    const shadowRoot = …

    shadowRoot.innerHTML = `
      <span>My Component</span>
    `;
  }
}
```

Scoped CSS

```
class MyComponent extends HTMLElement {
  constructor() {
    super();
    const shadowRoot = …

    shadowRoot.innerHTML = `
      <style></style>

      <span>My Component</span>
    `;
  }
}
```

Scoped CSS

```
class MyComponent extends HTMLElement {
  constructor() {
    super();
    const shadowRoot = …

    shadowRoot.innerHTML = `
      <style>
        span {
          font-weight: bold;
        }
      </style>

      <span>My Component</span>
    `;
  }
}
```

Scoped CSS

My Component

```
<my-component></my-component>
```

Scoped CSS

Regeln

Web Components' internes Markup lässt sich nicht von aussen stylen.

Web Components' eigene Styles „leaken" nicht nach aussen.

Scoped CSS


```
<my-component></my-component>

<style>
  my-component {
    border: 1px solid red;
  }
</style>
```

:host

…that’s all!

Wrap up „Web Components“

- 3x W3C Standards
- Templating
- Props
- Lifecycle
- Scoped CSS

…that’s all!

Wrap up „Web Components“

Doch wann können wir Web Components
produktiv einsetzen - inkl. IE11?

<u>…schon heute!</u>

Frameworks

Web Components schon heute!

- Polymer von Google, 2015
- Skate von Netlify, 2016
- Stencil von Ionic, 2017

im Fokus

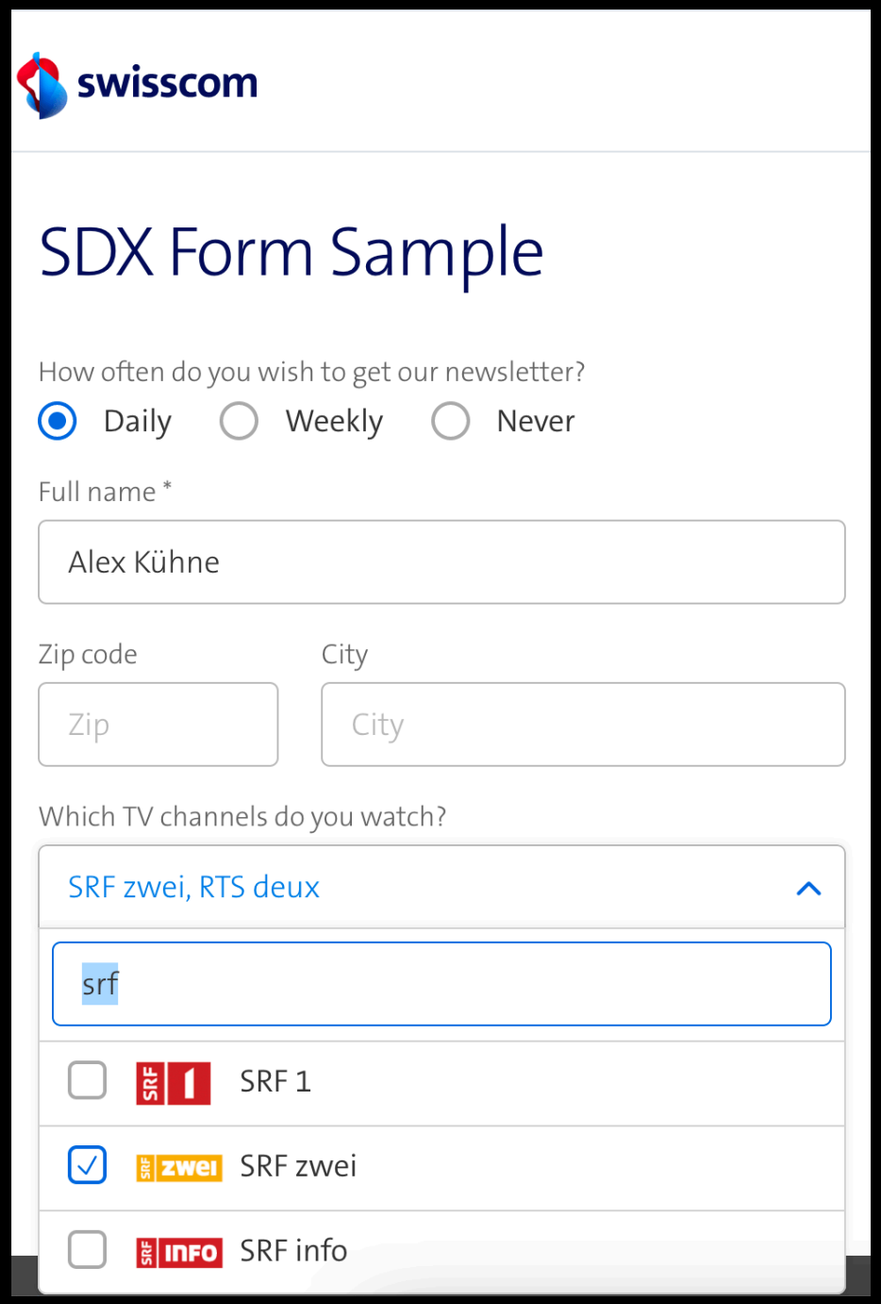
Stencil

Wer benutzt Stencil?

- SDX von Swisscom, 2018
- Apple Music von Apple, 2019
- Amazon Music von Amazon, 2020

Stencil

sdx.swisscom.com



Stencil

Library um Web Components zu bauen

sehr React-ähnlich

- TypeScript
- JSX und SASS
- Lazy loading, async rendering
- Testumgebung mit Jest und Puppeteer
- Polyfills für ältere Browser, z.B. IE11

Stencil

```
// my-counter.tsx
```

Stencil

```
// my-counter.tsx

export class MyCounter {
}
```

Stencil

```
// my-counter.tsx

export class MyCounter {
  @State() counter = 0;
}
```

Stencil

```
// my-counter.tsx

export class MyCounter {
  @State() counter = 0;
  @Prop() fontSize = 18;
}
```

Stencil

```
// my-counter.tsx

export class MyCounter {
  @State() counter = 0;
  @Prop() fontSize = 18;

  render() {
    return <div>{this.counter}</div>;
  }
}
```

Stencil

```
// my-counter.tsx

@Component()
export class MyCounter {
  @State() counter = 0;
  @Prop() fontSize = 18;

  render() {
    return <div>{this.counter}</div>;
  }
}
```

Stencil

```
// my-counter.tsx

@Component({
  tag: "my-counter"
})
export class MyCounter {
  @State() counter = 0;
  @Prop() fontSize = 18;

  render() {
    return <div>{this.counter}</div>;
  }
}
```

Stencil

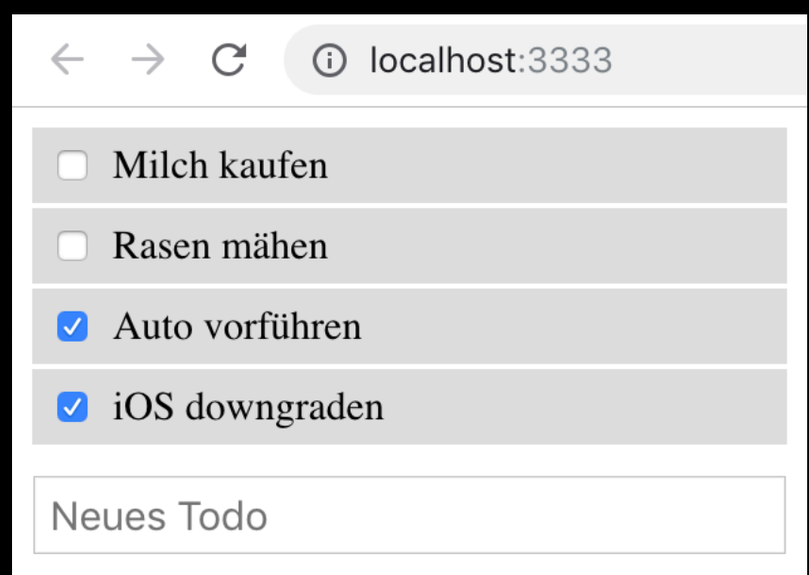
```
// my-counter.tsx

@Component({
  tag: "my-counter",
  styleUrl: "my-counter.css"
})
export class MyCounter {
  @State() counter = 0;
  @Prop() fontSize = 18;

  render() {
    return <div>{this.counter}</div>;
  }
}
```

Stencil

```
// my-counter.tsx

@Component({
  tag: "my-counter",
  styleUrl: "my-counter.css",
  shadow: true
})
export class MyCounter {
  @State() counter = 0;
  @Prop() fontSize = 18;

  render() {
    return <div>{this.counter}</div>;
  }
}
```

Praxisübung

`<my-todos>`



1. Todos anzeigen
2. als erledigt markieren
3. Hinzufügen
4. Erledigte zuunterst anzeigen
5. Integrationstest schreiben
   `npx stencil test --e2e`

Praxisübung

`<my-todos>`

Tipps 1/2

```
// Immutable Hinzufügen zu einem Array
myArray.concat({ key: "value" });

// Enter Key abfragen
if (e.key === "Enter") { … }
```

Praxisübung

<my-todos>

Tipps 2/2

```
// Sortierung
todos.sort((a, b) =>
  a.done === b.done ? 0 : (a.done ? 1 : -1)
);

// Eindeutiger Key für Loop, z.B. "name"
todos.map((todo) =>
  <div key={todo.name}></div>
);
```

Praxisübung

`<my-todos>`

Tipps für Integrationstest

```
// Elemente aus Shadow Root lesen
element.shadowRoot.querySelector(…);

// Input abfüllen und Enter simulieren
await page.evaluate(() =>
  input.value = …;
  input.focus();
);

await page.keyboard.press("Enter");
await page.waitForChanges();
```

Praxisübung

`<my-todos />`

Stencil Installation

```
$ git clone https://github.com/ionic-team/stencil-component-starter.git my-todos

$ cd my-todos

$ npm install

$ npm start
```

Auf `<my-component>` aufbauend arbeiten